Implement a CNN architecture to classify the MNIST handwritten dataset

Ihsan Md. Al-Aqib
ID: 18-37074-1

Abstract:

Computer vision aims to give computers the ability to see and analyze pictures in the same way that humans do. We'll learn how to build and train a basic Convolutional Neural Network (CNN) for classifying handwritten digits from a common dataset in this project. CNN is a deep learning approach for automatically categorizing input (after supplying the correct data), and it has mastered the art of classifying pictures for computers throughout the years. A typical dataset used in computer vision and deep learning is the MNIST handwritten digit classification challenge. Despite the fact that the dataset is successfully solved, it may be used to learn and practice how to build, analyze, and apply convolutional deep learning neural networks for image classification from the ground up. This includes how to create a difficult test harness for estimating the model's performance, how to investigate model modifications, and how to save and load the model to make predictions on real data. The objective of our project will be to develop a model that can more accurately recognize and determine a handwritten number from its picture. Using the ideas of Convolutional Neural Networks and the MNIST dataset, we hope to finish this.

Keywords: convolutional neural network; MNIST dataset; epoch; ReLu; softmax.

Introduction:

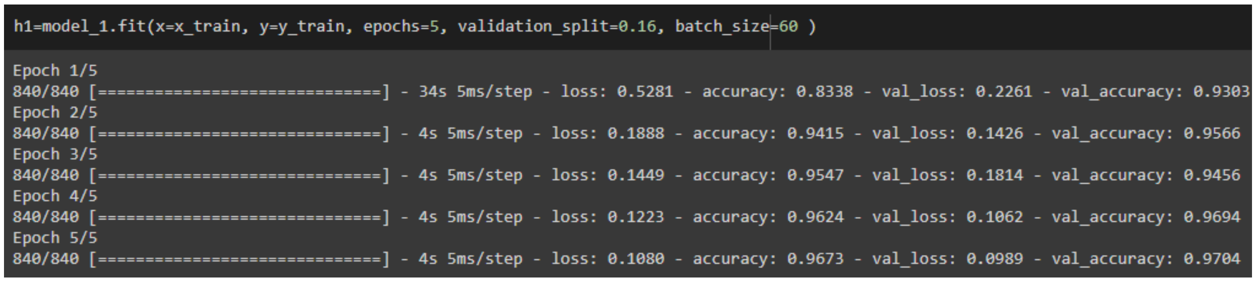
Convolutional Neural Networks (CNN) have established as one of the most appealing methodologies, and they have played a key role in a number of recent successes and difficult machine learning applications. As a result, we chose CNN for our difficult picture classification jobs. We may use it to recognize handwritten numbers, which is one of the most important academic and business operations. Handwriting digit recognition has a wide range of uses in our daily lives. We may use it at banks to read checks, organize exam paper file which has scan of hand written and a variety of other places.

MNIST database: The MNIST (Modified National Institute of Standards and Technology) database is a collection of handwritten digits. It may be used to train different image processing systems. In the field of machine learning, the database is also commonly utilized for training and testing. There are 60,000 training and 10,000 testing examples in this package. Each picture has a predetermined size. The photos are 28*28 pixels in size. It's a database for those who wish to experiment with machine learning and pattern recognition algorithms on real-world data with little preparation and formatting. This database will be used in our experiment.

Convolutional Neural Networks: Deep artificial neural networks are convolutional neural networks. It may be used to recognize faces, people, street signs, cancers, platypuses, and a variety of other visual data. A CNN's main building piece is the convolutional layer. The parameters of the layer are a series of learnable filters/kernels with a narrow receptive field but that span the whole depth of the input volume. Each filter is convolved over the width and height of the input volume during the forward pass, computing the dot product and providing a 2-dimensional activation map of that filter. As a consequence, the network learns when it sees a certain sort of feature at a given geographical location.

Results:

Model_1 Test with optimizer (Adam)



```
h1=model_1.fit(x=x_train, y=y_train, epochs=5, validation_split=0.16, batch_size=60 )

Epoch 1/5
840/840 [==============================] - 34s 5ms/step - loss: 0.5281 - accuracy: 0.8338 - val_loss: 0.2261 - val_accuracy: 0.9303
Epoch 2/5
840/840 [==============================] - 4s 5ms/step - loss: 0.1888 - accuracy: 0.9415 - val_loss: 0.1426 - val_accuracy: 0.9566
Epoch 3/5
840/840 [==============================] - 4s 5ms/step - loss: 0.1449 - accuracy: 0.9547 - val_loss: 0.1814 - val_accuracy: 0.9456
Epoch 4/5
840/840 [==============================] - 4s 5ms/step - loss: 0.1223 - accuracy: 0.9624 - val_loss: 0.1062 - val_accuracy: 0.9694
Epoch 5/5
840/840 [==============================] - 4s 5ms/step - loss: 0.1080 - accuracy: 0.9673 - val_loss: 0.0989 - val_accuracy: 0.9704
```

Fig: model_1 result

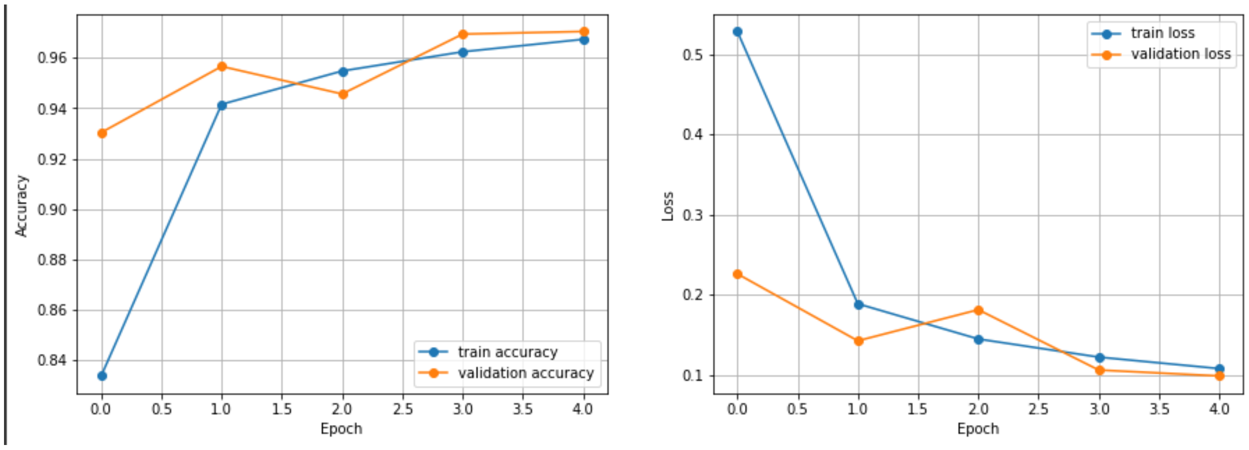
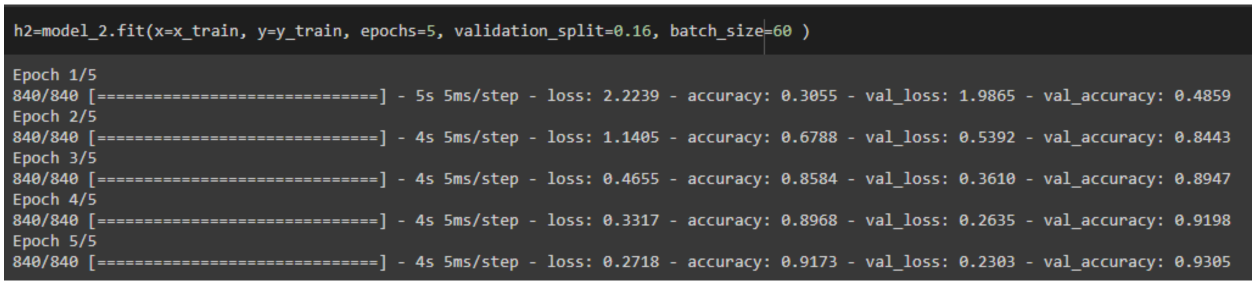


Fig: model_1 result graph

Model_2 Test with optimizer (SGD)



```
h2=model_2.fit(x=x_train, y=y_train, epochs=5, validation_split=0.16, batch_size=60 )

Epoch 1/5
840/840 [==============================] - 5s 5ms/step - loss: 2.2239 - accuracy: 0.3055 - val_loss: 1.9865 - val_accuracy: 0.4859
Epoch 2/5
840/840 [==============================] - 4s 5ms/step - loss: 1.1405 - accuracy: 0.6788 - val_loss: 0.5392 - val_accuracy: 0.8443
Epoch 3/5
840/840 [==============================] - 4s 5ms/step - loss: 0.4655 - accuracy: 0.8584 - val_loss: 0.3610 - val_accuracy: 0.8947
Epoch 4/5
840/840 [==============================] - 4s 5ms/step - loss: 0.3317 - accuracy: 0.8968 - val_loss: 0.2635 - val_accuracy: 0.9198
Epoch 5/5
840/840 [==============================] - 4s 5ms/step - loss: 0.2718 - accuracy: 0.9173 - val_loss: 0.2303 - val_accuracy: 0.9305
```

Fig: model_2 result

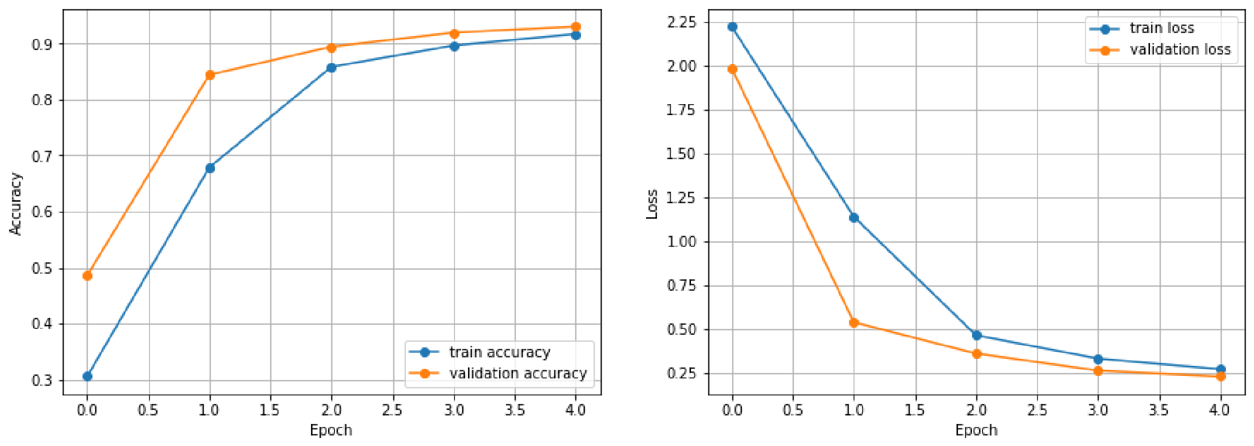
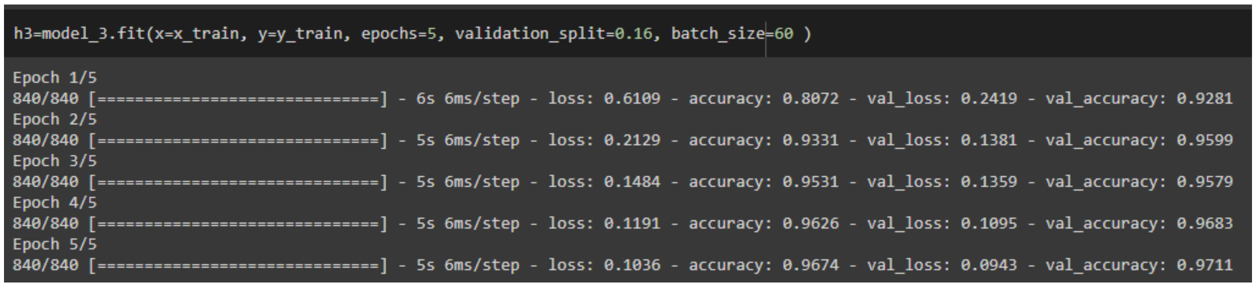


Fig: model_2 result graph

Model_3 Test with optimizer (RSMProp)



```
h3=model_3.fit(x=x_train, y=y_train, epochs=5, validation_split=0.16, batch_size=60 )

Epoch 1/5
840/840 [==============================] - 6s 6ms/step - loss: 0.6109 - accuracy: 0.8072 - val_loss: 0.2419 - val_accuracy: 0.9281
Epoch 2/5
840/840 [==============================] - 5s 6ms/step - loss: 0.2129 - accuracy: 0.9331 - val_loss: 0.1381 - val_accuracy: 0.9599
Epoch 3/5
840/840 [==============================] - 5s 6ms/step - loss: 0.1484 - accuracy: 0.9531 - val_loss: 0.1359 - val_accuracy: 0.9579
Epoch 4/5
840/840 [==============================] - 5s 6ms/step - loss: 0.1191 - accuracy: 0.9626 - val_loss: 0.1095 - val_accuracy: 0.9683
Epoch 5/5
840/840 [==============================] - 5s 6ms/step - loss: 0.1036 - accuracy: 0.9674 - val_loss: 0.0943 - val_accuracy: 0.9711
```

Fig: model_3 result

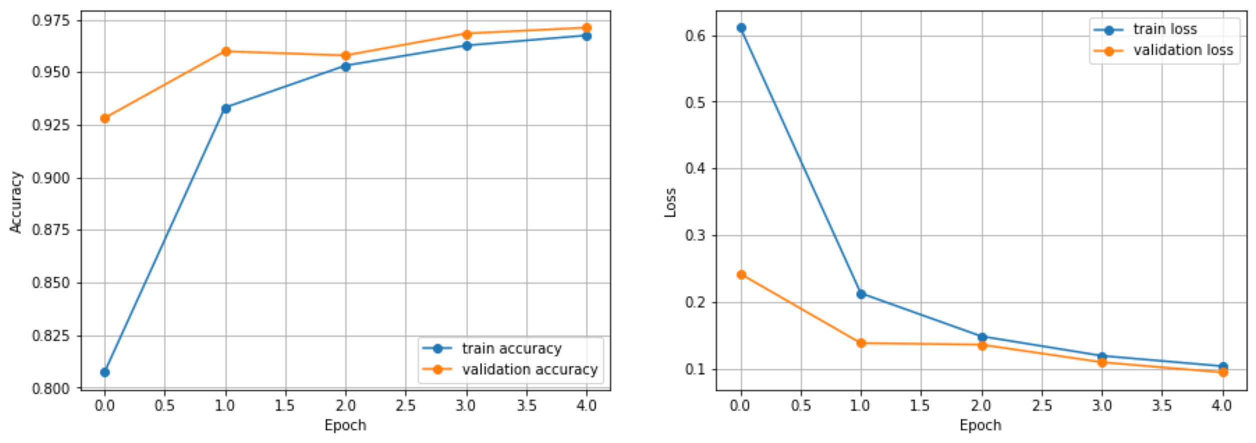


Fig: model_3 result graph

Discussion:

Convolutional layer, pooling layer(s), and fully connected layer are the three basic layers of a CNN model.

I. **Convolutional Layer:** This layer extracts high-level input features from input data and sends them along as feature mappings to the next layer.
II. **Pooling Layer:** It is used to reduce the dimensions of data by applying pooling on the feature map to generate new feature maps with reduced dimensions. PL takes either maximum or average in the old feature map within a given stride.
III. **Fully-Connected Layer:** Finally, the task of classification is done by the FC layer. Probability scores are calculated for each class label by a popular activation function called the softmax function.

Model Name	**Optimizer**	**No. of epochs**	**Accuracy**
Model_1	Adam	5	97.04%
Model_2	SGD	5	93.05%
Model_3	RSMProp	5	97.11%

Table1: Comparing model accuracy

After five epochs, In Model_3 loss 9.43, corresponding to a 97.11 percent identification rate as shown in fig:Model_3 result which is the highest accuracy among those model. For such a basic model with CPU training and shorter training time, the results are rather good (about 20 minutes).